НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації і управління

«**До захисту допущено**»
В.о. завідувача кафедри
_____ __*О.А.Павлов*__
(підпис) (ініціали, прізвище)
“___” _____ 2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки *6.050103 «Програмна інженерія»*

спеціальність *«Програмне забезпечення систем»*

на тему: *Серверна частина інтернету речей*

Виконав: студент 4 курсу, групи ІП-52

Сухоруков Михайло Кирилович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *старший викладач Олійник Ю.О.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант з графічної документації *доц. к.т.н. Ліщук К.І.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент *проф. каф. ОТ, д.т.н., Сімоненко В.П.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

АНОТАЦІЯ

Дана робота присвячена дослідженню серверних систем, які забезпечують доступ до Інтернету для речей, а також досвід хмарної платформи.

Під час огляду ми розробляємо платформу для новітніх нових підходів до перегляду коренів прийому, зберігання та відстеження великих обсягів даних. Для розробки платформ, ви отримаєте копію інформаційних бюлетенів і відкриту базу даних. Випали на платформу, трясучись у хмарному виборі. Забезпечує архітектуру платформи, що забезпечує її використання і масштабованість, а також дозволяє реалізувати рішення для моніторингу мережевих принтерів і обробки адміністративних даних, які є більш функціональними, ніж іконки.

Платформа попередньо встановлена на прикладі. Вказується також, що підключення реальної системи до системи також вказано.

Загальний оборот: 60 найменувань, 5 рисунків, 19 таблиць, 32 посилання, 1 додаток на 12 сторінках.

КЛЮЧОВІ СЛОВА: РЕЧІ В ІНТЕРНЕТІ, ХМАРНА ПЛАТФОРМА, СИСТЕМНЕ ВІДОБРАЖЕННЯ, ВЕЛИКІ ПАКЕТНІ ДАНІ, МОНІТОРИНГ МЕРЕЖІ, APACHE SPARK, APACHE KAFKA, APACHE CASSANDRA, MONGODB.

ABSTRACT

In the dissertation are investigated server systems of Internet things and directed on development of a cloud platform.

The article describes the use of new approaches to the development of distributed systems that receive, store and process large volumes of data. The platform was built using modern libraries and open source databases. The cloud of deployment of the system is considered. The dissertation offers a platform architecture that ensures its high scalability and efficiency, as well as allows you to implement solutions for monitoring the network of devices and management of administrative data that are better than those provided by existing systems.

The application of the platform has been demonstrated on an example. He also showed the connection of the real device to the system.

The thesis contains 60 pages, 5 figures, 19 tables, 32 references and 1 supplement on 12 pages.

KEYWORDS: INTERNET OF THINGS, CLOUD PLATFORM, DISTRIBUTION SYSTEM, MAXIMUM DATA PROCESSING, NETWORK MONITORING APAKE, APACHE KAFKA, APACHE KASANDRA, MONGODB.

Пояснювальна записка
до дипломного проекту

на тему: Серверне застосування для Інтернету речей

Київ – 2019 року

ЗМІСТ

ВСТУП

Інтернет речей швидко стає основою всього нашого життя, оскільки повсякденні пристрої в будинках споживачів і на підприємствах підключені до Інтернету і генерують дані.

IoT - це концепція, що розвивається, і вона також стає все більш складним організмом, оскільки більше пристроїв - і більше видів пристроїв - створюються з підключенням, запеченим з самого початку.

В цілому, IoT - це органічний розвиток технології, який продовжує поширюватися по всьому світу і стосується більшої кількості аспектів життя споживачів і бізнесу, з якими вони взаємодіють.

Існує безліч реальних додатків Інтернету речей, починаючи від споживчого інтернету (IoT) і корпоративного Інтернету (IoT) до виробництва та промислових IT (IIoT). Застосування IoT охоплює численні вертикалі, включаючи автомобільну, телекомунікаційну, енергетичну та багато іншого.

Носійні пристрої з датчиками та програмним забезпеченням можуть збирати та аналізувати дані користувача, посилаючи повідомлення іншим користувачам про технології з метою полегшення та зручності життя користувачів. Носінні пристрої також використовуються для забезпечення громадської безпеки - наприклад, поліпшення часу реагування перших респондентів під час надзвичайних ситуацій шляхом надання оптимізованих маршрутів до місця розташування або відстеження життєвих ознак будівельних робітників або пожежників на небезпечних для життя місцях.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Змістовний опис і аналіз предметної області

У цій роботі метод FIA використовується для проведення техніко-економічного обґрунтування розробки платформи для Інтернету речей, заснованої на хмарі. Основні конструктивні рішення визначаються відповідно до вимог всієї системи, тому її компоненти підсистеми повинні задовольняти їх. Тому доречно аналізувати функції програмного продукту, призначеного для отримання, обробки та зберігання даних з пристроїв в Інтернеті.

Відповідно, варто вибрати систему показників якості програмного продукту.

Технічні вимоги до виробу такі:

- Програмний продукт повинен функціонувати на товарному обладнанні зі стандартним набором компонентів і загальним в поточних характеристиках ЦОД;
- Забезпечити високу швидкість обробки великих обсягів даних, як в режимі реального часу (потокова обробка), так і для виконання аналітичних запитів;
- Забезпечити зручність та зручність взаємодії з користувачами системи, а також з розробниками програмного забезпечення, якщо такі є.
- Писати заявку на платформу або інтегрувати її в інші системи конкретного підприємства;
- Забезпечення мінімальних витрат на впровадження програмного продукту.

Індикатором успіху проекту є робоча система, яка:

- Вирішує основні завдання таких платформ (зберігання даних, аналіз даних тощо), тобто впровадження цих підсистем з урахуванням загальних підходів до розвитку розподілених систем обробки даних;
- Розроблено з урахуванням передового досвіду та підходів до архітектури програмного забезпечення [8-13], які дозволяють забезпечити функціональні та нефункціональні вимоги до системи;
- Має певну функціональність, яку інші системи не мають (наприклад, надійну доставку повідомлень між пристроями та адміністраторами та пристроями керування пристроями);
- Містить засоби для розгортання хост-систем у хмарі, а також інструменти для управління групою серверів.

1.2 Аналіз успішних IT-проектів

Наразі існує декілька платформ, що працюють в цій галузі. Найбільш відомі з них: Amazon Web Services IoT Platform, Google Cloud IoT Platform, ThingWorx, Oracle IoT, IBM IoT Platform.

Всі вищезазначені рішення забезпечують загальну функціональну дію, що включає в себе зберігання даних, а також широкий спосіб їх відображення та аналізу. Перенесення платформи - це оголошення з замкнутим джерелом, яке є їх значним ненульовим значенням. Таке закриття передбачає їх розширення, адаптацію та оптимізацію відповідно до конкретного сценарію ротації. Протестуючі, які використовують платформи, що працюють на блокуванні такого постачальника, не підлягають повторній адаптації до інших платформ без значних витрат на шасі та обладнання.

Навряд чи існує платформа з відкритим кодом. Ви можете протестувати функціональну платформу, яка дозволить вам створювати списки відкритих джерел і бібліотеки, оскільки часто існує безліч ліцензій з

відкритим кодом, які можуть бути процитовані системою і які вже застаріли і мають високий ступінь гнучкості.

Розглянемо, наприклад, одну з найбільш функціональних і популярних існуючих платформ - платформу AWS IoT.

Основні функції для роботи з системою:

- SDK для пристрою, який надсилає повідомлення.
- Авторизувати та авторизувати використання сертифікатів SigV4 або X.509.
- Шлюзові пристрої, які здійснюють зв'язок на базі pub / sub через протоколи MQTT, WebSocket, HTTP 1.1.
- Реєстрація для зберігання інформації про пристрій.
- Device Shadows - Механізм зберігання останнього прийнятого пристрою для подальшого запису через REST API.
- Механізм визначення правил, що описують інтеграцію з послугами AWS.

Як ви можете бачити, AWS IoT пропонує дуже просунуті функції безпеки, але дозволяє збільшити серйозність AWS за допомогою унікального протоколу MQTT і не надає API для визначення власних ланцюгів обробки даних.

Огляд іншої платформи для Інтернету - IBM IoT Platform. Розробники розділили його функціональність на чотири основні блоки:

1) З'єднання. Платформа пропонує багато SDK для пристроїв і шлюзів, а також API для взаємодії клієнтського програмного забезпечення. Ці функції доступні для різних мовних програм, включаючи оптимізовані низькорівневі рішення для вбудованих систем C. Цей функціональний блок призначений для надійного підключення пристроїв до платформи даних. Основними протоколами передачі є MQTT і HTTPS.

2) Управління ризиками. Ця група відповідає за управління великими групами пристроїв, що дозволяє користувачам переглядати інформацію про свою роботу в панелях керування. Існують також адміністративні звіти про ненормальні ситуації, які дозволяють швидко ізолювати різні випадки.
3) Керуйте інформацією. Блок функціональності, що дозволяє керувати збереженням і архітектурою оперативної інформації і метаданих. Доступні варіанти аналізу та конвертації даних, генерування звітів. Також можна вводити дані з різних джерел і платформ.
4) Аналітика. Інтеграція з IBM Watson дозволяє користувачеві виконувати складську аналітику, використовуючи існуючі карти Watson. Наприклад, блок аналізу природної мови, різноманітні алгоритми машинного навчання, відеоаналіз і аналіз зображень, аналіз текстових даних.

IBM IoT Platform, яка має великі можливості аналізу даних, реалізовані у IBM Watson. Інша функціональність досить характерна для таких платформ. Основним недоліком є закритість платформ, а також дотримання неможливості його розширення.

Іншою функціональною і потужною платформою є Oracle IoT. Розглянемо більш детально його можливості, які умовно розділені на три частини:

1) З'єднання. Цей пристрій має такі функції:

- Віртуалізація пристроїв. Кожен пристрій являє собою як набір ресурсів, так і повну абстракцію пристроїв підключення та стандартів інтеграції.
- Надійне спілкування. Існує надійний, двосторонній зв'язок, що забезпечує доставку повідомлень через неточні мережі і до / з пристроїв, які працюють і підключаються до мережі лише час від часу.

- Гнучка топологія. Пристрої можуть бути підключені до платформи, використовуючи різні топології мережі, такі як бібліотека клієнтського програмного забезпечення або безпосередньо через REST API. Доступні бібліотеки пристроїв на Java SE, Java ME, POSIX C. Для шлюзів, не протоколи IP, такі як підтримка Bluetooth, Z-Wave, Modbus і OPC.

2) Аналіз. Надає такі функції:

- Обробка потоку. Аналіз даних агрегації, фільтрації та подій в реальному часі.
- Підвищення даних. Можливість збагачувати потоки необроблених даних контекстною інформацією, наприклад, пристрої метаданих і компілюються потоки даних.
- Сховище подій. Запропоновані потоки даних можуть бути накладені на індексовані сховища і служби, які є частиною Oracle Cloud.

3) Інтеграція. Має такі функції:

- Підключення підприємств. Дані IoT можуть бути відправлені в інше програмне забезпечення, яке керує процесорами та Oracle Cloud.
- REST API. Забезпечує інтеграцію з іншими системами, як в Oracle Cloud, так і за ним.
- Команди пристроїв та їх керування. Можна надсилати повідомлення користувачам, незалежно від того, чи підключено ви до цієї мережі.

Як бачите, переваги Oracle IoT мають широкі можливості для передачі даних і пристроїв, а також підтримку різних протоколів (функцій шлюзу). Однак, використовуйте більшість користувачів послуг, які використовують систему Oracle Cloud.

1.3 Аналіз вимог до програмного забезпечення

Система повинна містити такі типи користувачів як адміністратор і користувач.

Адміністратор – це користувач з особливими правами, має доступ до сторонніх джерел інформації для завантаження даних з них до бази даних системи.

Користувач має доступ до функціоналу сервісу: може експортувати дані, переглядати візуалізації аналітичних показників, розраховувати вартість конвертації різних валют одна між одною.

В системі повинні бути реалізовані наступні функції:

- Перегляд візуалізацій аналітичних показників;
- Статистичний аналіз курсу валюти;
- Експорт інформації про курс валют до файлу;
- Розрахунок вартості конвертації між валютами.

1.3.1 Розроблення функціональних вимог

Схема структурна варіантів використання наведена у графічному матеріалі.

В системі передбачено наступні варіанти використання, описані у таблицях 1.1 – 1.19:

Варіанти використання в системі представлені нижче:

Таблиця 1.1 – Варіант використання UC001

Назва	Отримання токену автентифікації
Опис	Використовується всіма користувачами системи для отримання токену автентифікації за вказаними даними доступу
Учасники	Будь-хто
Передумови	Користувач не зареєстрований у системі

Постумови	Користувач проходить авторизацію
Основний сценарій	Користувач відправляє дані та отримує токен входу у систему

Таблиця 1.2 – Варіант використання UC002

Назва	Додавання нового пристрою
Опис	Користувач має можливість додавати новий пристрій до системи
Учасники	Користувач
Передумови	Пристрій не зареєстрований у системі
Постумови	Пристрій зареєстрований у системі
Основний сценарій	Система отримує запит на реєстрування пристрою за вказаними параметрами и реєструє його

Таблиця 1.3 – Варіант використання UC003

Назва	Отримання даних про пристрій
Опис	Повертає дані про пристрій з указаним id (розташування, тип тощо)
Учасники	Користувач
Передумови	Потрібно отримати дані про пристрій
Постумови	Дані про пристрій повертаються з серверу

Основний сценарій	За вказаним id надсилається запит на отримання даних про пристрій

Таблиця 1.4 – Варіант використання UC004

Назва	Змінення даних про пристрій
Опис	Користувач має можливість змінювати дані про пристрій за указаним id
Учасники	Користувач
Передумови	Пристрій має застарілі дані
Постумови	Пристрій має оновлені дані
Основний сценарій	Система отримує запит на зміну даних пристрою за вказаними параметрами і оновлює його

Таблиця 1.5 – Варіант використання UC005

Назва	Реєстрація нового користувача
Опис	Додавання нового користувача системи (адміністратора, аналітика тощо).
Учасники	Користувач
Передумови	Користувач не зареєстрований у системі

Постумови	Користувач зареєстрований у системі
Основний сценарій	Система отримує запит на реєстрацію нового користувача за вказаними параметрами і реєструє його

Таблиця 1.6 – Варіант використання UC006

Назва	Отримання даних про користувача
Опис	Повертає дані про користувача з указаним id (розташування, тип тощо)
Учасники	Користувач
Передумови	Потрібно отримати дані про користувача
Постумови	Дані про користувача повертаються з серверу
Основний сценарій	За вказаним id надсилається запит на отримання даних про користувача

Таблиця 1.7 – Варіант використання UC007

Назва	Змінення даних про користувача
Опис	Користувач має можливість змінювати дані про користувача за указаним id
Учасники	Користувач

Передумови	Користувача має застарілі дані
Постумови	Користувача має оновлені дані
Основний сценарій	Система отримує запит на зміну даних користувача за вказаними параметрами і оновлює його

Таблиця 1.8 – Варіант використання UC008

Назва	Считування даних зі вказаного пристрою
Опис	Повертає дані зі вказаного пристрою, оброблені відповідним чином.
Учасники	Користувач
Передумови	Потрібно отримати інформацію з пристрою
Постумови	Дані з пристрою отримані
Основний сценарій	Система отримує запит на отримання даних з пристрою за вказаним id

Таблиця 1.9 – Варіант використання UC009

Назва	Запис команди до пристрою
Опис	Викликається адміністратором для запису команди для пристрою

Учасники	Адміністратор
Передумови	Потрібно записати команду на пристрій
Постумови	Команда для пристрою записана
Основний сценарій	Система отримує запит на запис команди для пристрою

Таблиця 1.10 – Варіант використання UC011

Назва	Повертає актуальні команди
Опис	Повертає актуальні команди, призначені вказаному пристрою
Учасники	Користувач
Передумови	Потрібно отримати усі команди по конкретному пристрою
Постумови	Усі команди по конкретному пристрою отримані
Основний сценарій	Система отримує запит на отримання усіх команд по конкретному пристрою та повертає їх

Функціональні вимоги додатку описано наступними таблицями.

Таблиця 1.11 – Опис функціональної вимоги REQ001

						Арк.
Зм.	Арк.	№ докум.	Підпис	Дата	КПІ.ІП-5214.045490.02.81	19

Поме REQ001

p	
Назва	Реєстрація користувачів в системі
Опис	Система надає можливість користувачу реєструватись в системі

Таблиця 1.12 – Опис функціональної вимоги REQ002

Номер	REQ002
Назва	Реєстрація пристроїв в системі
Опис	Система надає можливість зареєстрованому користувачу реєструвати пристрої

Таблиця 1.13 – Опис функціональної вимоги REQ003

Номер	REQ003
Назва	Збереження даних у чергах
Опис	Система надає можливість пристроям зберігати данні черзі повідомлень

Таблиця 1.14 – Опис функціональної вимоги REQ004

Номер	REQ004
Назва	Отримання данних про пристрій
Опис	Система надає можливість користувачу отримати данні про пристрій

Таблиця 1.15 – Опис функціональної вимоги REQ005

Номер	REQ005
Назва	Отримання даних про користувача
Опис	Система надає можливість отримати дані про користувача

Таблиця 1.16 – Опис функціональної вимоги REQ006

Номер	REQ006
Назва	Отримання даних з пристрою
Опис	Система надає можливість отримати дані зі вказаного пристрою, оброблені відповідним чином

Таблиця 1.17 – Опис функціональної вимоги REQ007

Номер	REQ007
Назва	Запис команди до пристрою
Опис	Система дозволяє адміністратору записувати команди для пристрою

Таблиця 1.18 – Опис функціональної вимоги REQ008

Номер	REQ008
Назва	Отримання актуальних команд
Опис	Система дозволяє користувачу отримувати актуальні команди, призначені вказаному пристрою

Таблиця 1.19 – Опис функціональної вимоги REQ009

Номер	REQ009
Назва	Оновлення статусних повідомлень
Опис	Система дозволяє користувачу оновити статусне повідомлення зі вказаним id, наприклад, для запису відмітки про його перегляд адміністратором

Взаємозв’язки між вимогами клієнтського застосунку відображені на рисунку 1.1.

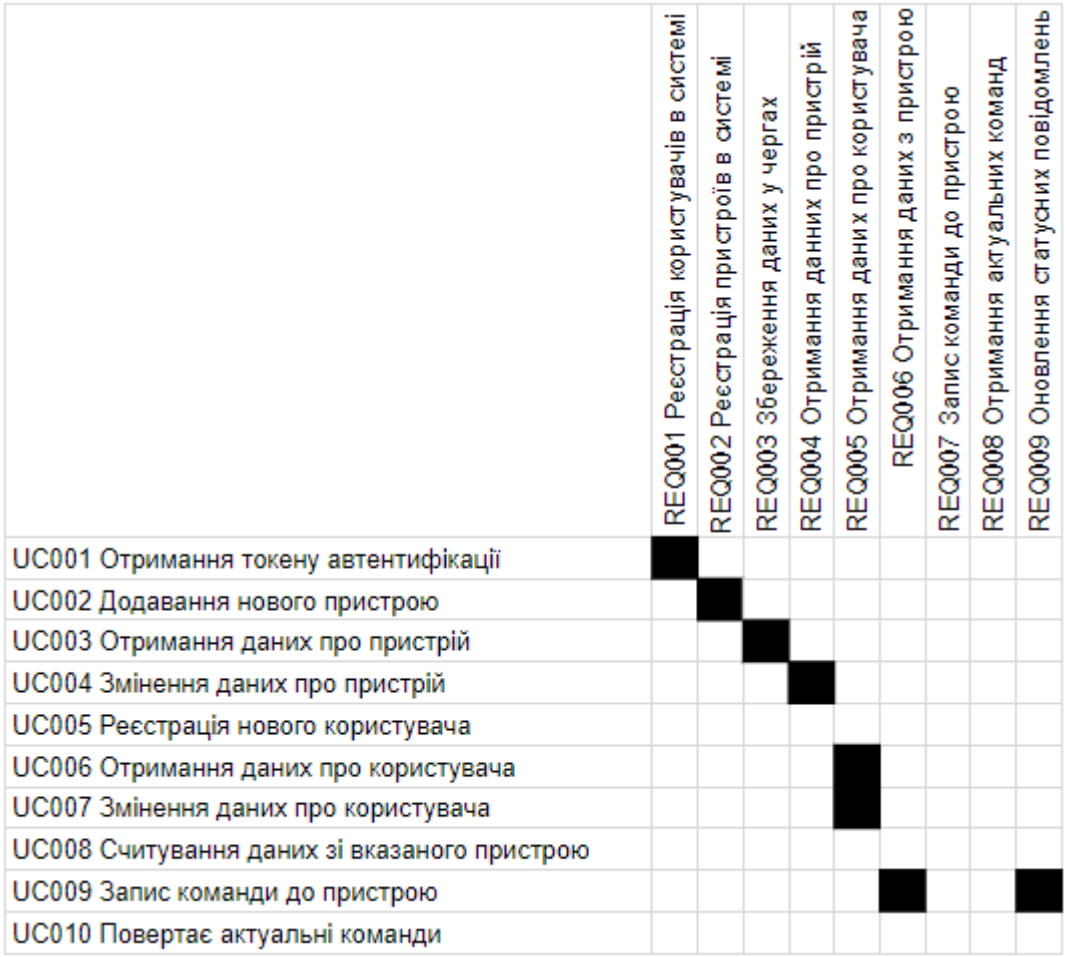

Рисунок 1.1 - Матриця залежності між вимогами застосунку і варіантами використання

1.1 Висновки по розділу

У цьому розділі було описано та проаналізовано предметну область розробки. Було виділено успішні IT- проекти у даній області та виконано порівняння даного проекту с готовими продуктами.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Архітектура системи

Опишемо оцінені значення комбо, разом з ними, а також напишемо і виведемо їх. Це зображення знаходиться на діаграмі (малюнок 2.1). На діаграмах негайно вказується активність певних об'єктів і бібліотек (наприклад, баз даних), які будуть використовуватися в тому чи іншому компоненті. Їх вибір буде обґрунтовано на пізніх етапах.

Сайт з відкритим вихідним кодом ("Джерела даних") має величезний вигляд, а також заміряє температуру, і її викликає хтось інший. Досвідчений є публічним фронтовим сервером. Розділ цих сертифікатів передає інформацію аутентифікації та інформацію власника (наприклад, ідентифікатор, для якого він може бути розпізнаний).

Передні елементи керування контролюють інформацію автентифікації. А. Для цих даних, у фронтальних церквах, функція передачі даних гіпецеризацією визначає тип розширення і підтверджує тему (предмет) в темряві (Apache Kafka). Крім того, передні кінцеві перевірки перевіряють цілісність даних (виконують зварювальник).

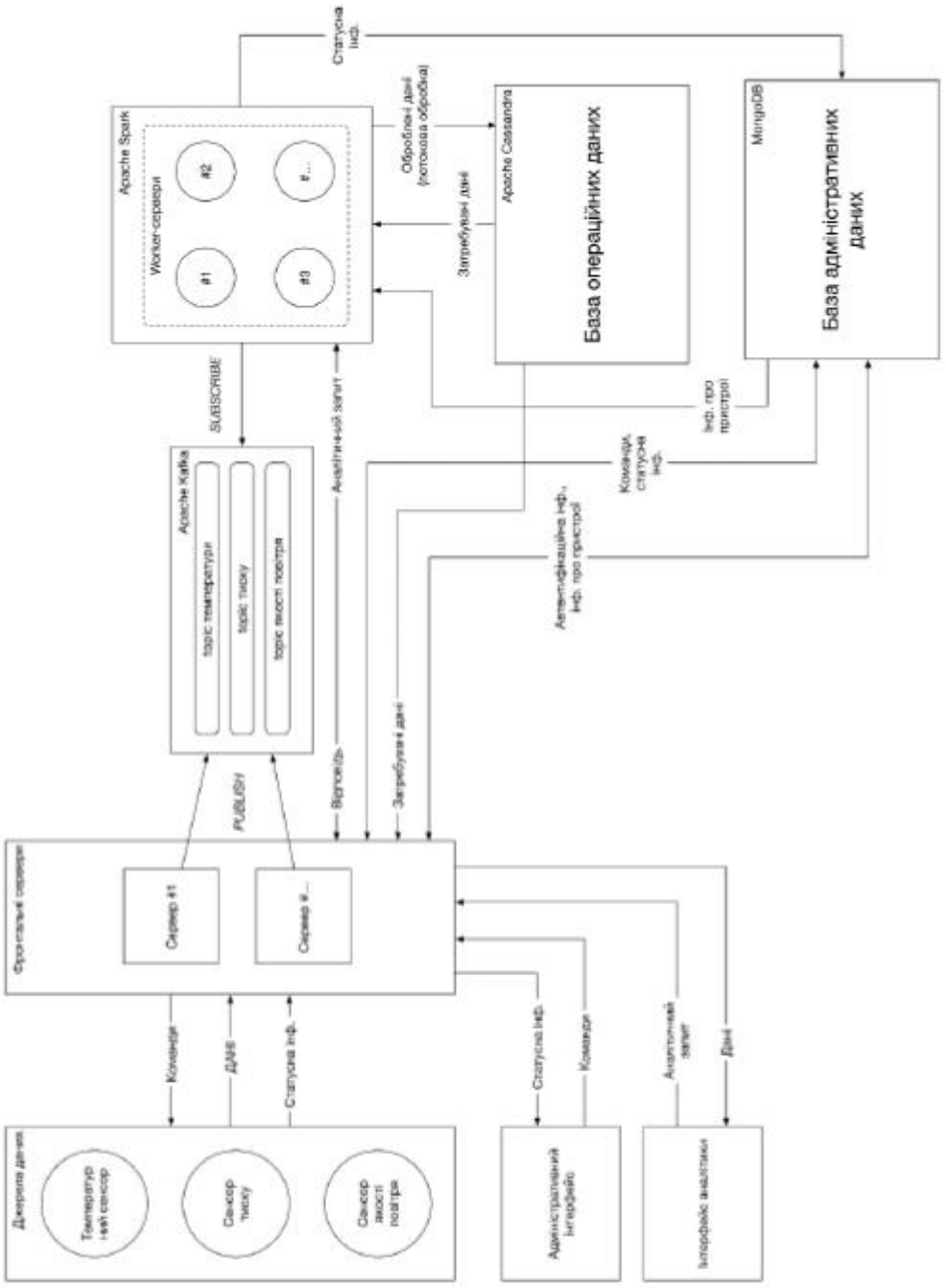

Рисунок 2.1 – Запропонована архітектура платформидля Інтернету речей

Черга пакетів виконується шаблоном реєстрації для підписки на публікацію. Сервери публікують дані в Apache Kafka.

На стороні підписки за обробку даних відповідальна Apache Spark. Він завантажує первинний журнал даних, який зберігається в робочій базі даних. Перевірка даних на даному етапі може бути або мікро-партією, або мікропакетною. Мікропроцесор може також емулювати повну дужку міхура, що дозволяє вставляти обладнання та багаж упаковки. Крім того, мікропроцесорне відставання повертає журнали агрегації, і журнали вікон, наприклад, повинні зібрати певну кількість підказок за хвилину, знайти їх значення і зберегти в базі даних. Також можна виконувати фільтр (або виявляти аномалії) на рівні мікропакетів.

База даних, в якій надаються операційні дані, забезпечує надійне зберігання збережених даних. Нижню частину екрану дуже складно оптимізувати після запрошення, і операція читання, очевидно, набагато менше. Попередня черга перевіряє буфери випадковим чином, якщо база даних не здатна до робочих даних (наприклад, завантаження навантаження) для збереження даних зі швидкістю їх ремонту. Зверніть увагу, що ця база даних не вимагає таких значних гарантій, які надаються традиційними базами даних транзакцій.

Наразі буде описано послідовність проходження інформації від пристроїв до постійного сховища. Наочно потоки даних в цьому сценарії використання показано на діаграмі (рис. 2.2).

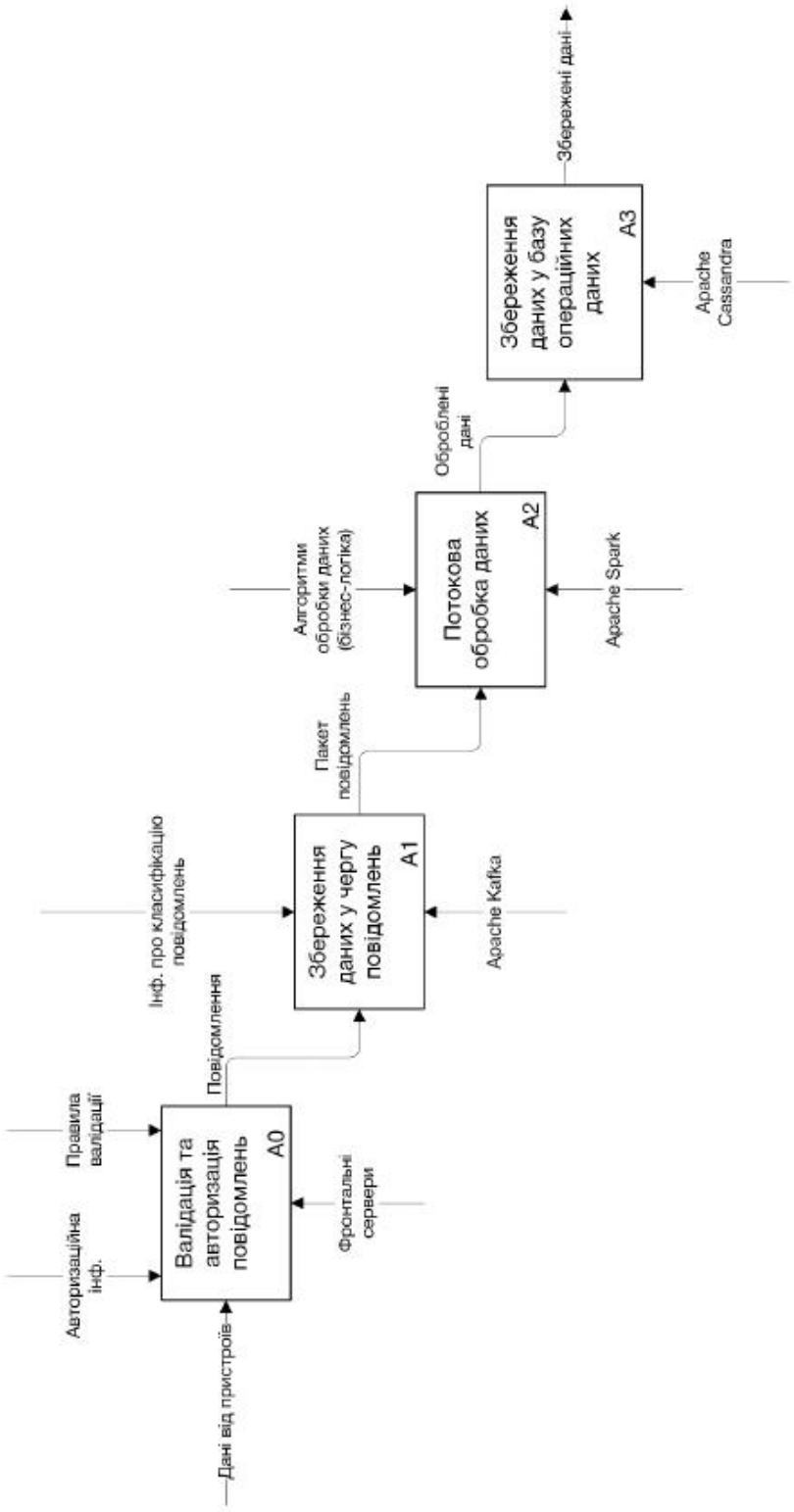

Рисунок 2.2 –Діаграма потоків даних і процесів при отриманні даних від сенсорів

Опишемо передачу інформації від пристрою до адміністратора. Такою інформацією може бути певна статусна інформація, скажімо, стан акумуляторів пристрою чи певні попередження про неполадки.

Ця інформація повинна переходити з фронтальних серверів на сервер, який, як і з отриманими операційними даними, завантажує його і видаляє аутентифікацію адреси запиту. З цієї причини генеруються дані для бази даних адміністративних даних.

У цьому випадку, як правило, існують значні гарантії для безпечного зберігання та обміну даними. З цієї причини, оскільки сервер інтерфейсу передає дані до бази даних, він шукає підтвердження їх збереження та збереження, перш ніж він стане підтвердженням власності, яке може забезпечити виконання певної судової справи у випадку, якщо інсайдерське спотворення є важливим.

Щоб зберегти цю інформацію, адміністратор просить довірену особу перевірити свої права на ці дані і запитує запит до бази даних за відсутності запиту предиката (наприклад, посилання на вказаний ідентифікатор).

Дані для цього випадку наведені на діаграмі (рис. 2.3).

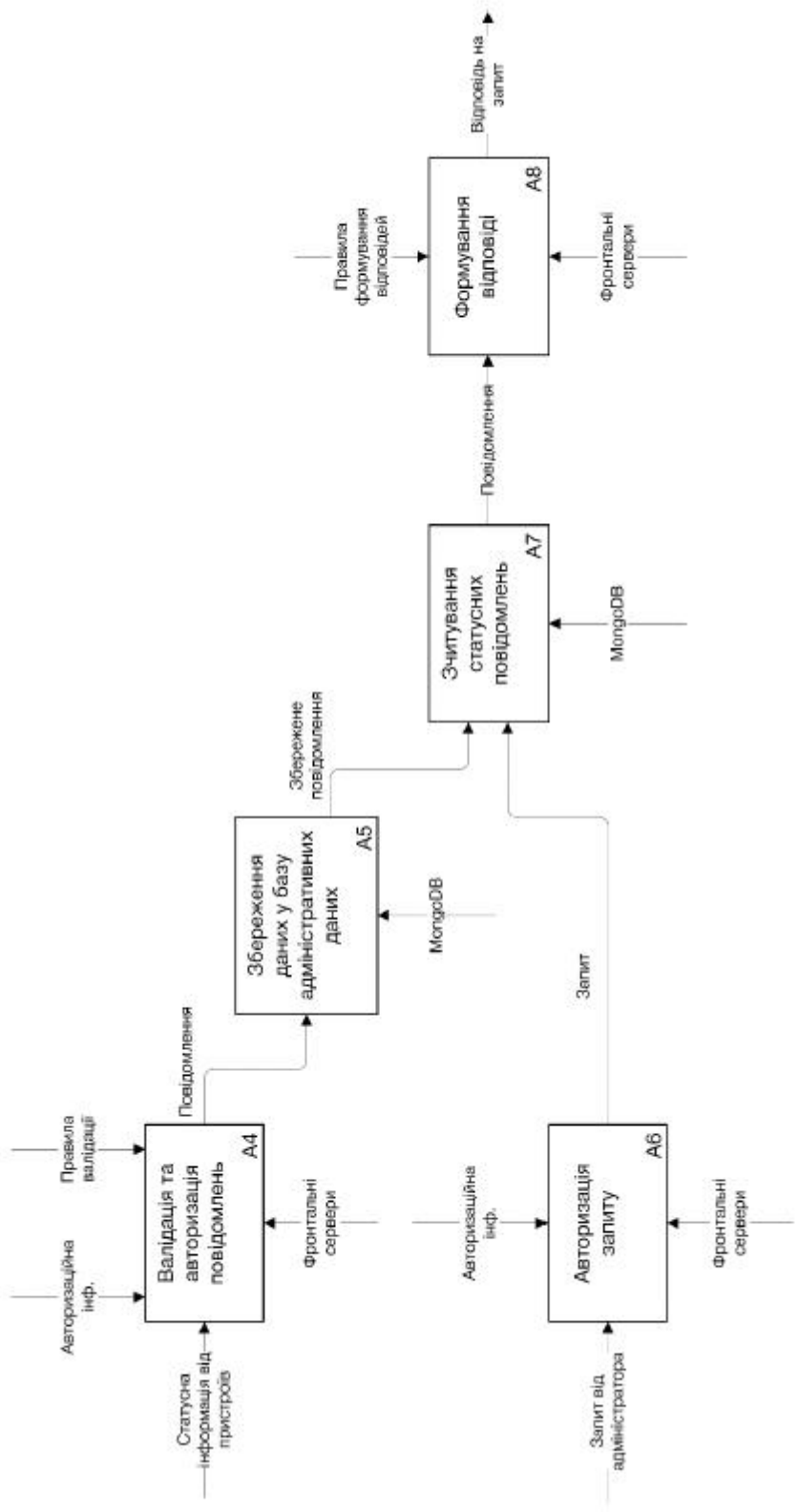

Рисунок 2.3 –Діаграма потоків даних і процесів при видсілланні інформації про статуси від пристрою до адміністратора

Іще одним джерелом статусної інформації є механізми моніторингу. Адміністратори у вигляді правил задають певні умови, які мають виконуватися при очікуваній роботі пристрою, та визначають порядок їх перевірки.

Наприклад, адміністратор може визначити правило, що певний пристрій має надсилати не менше деякої кількості повідомлень за одиницю часу. У разі, якщо ця умова не виконується, тобто спрацьовує певний тригер, платформа надсилає статусне повідомлення, аби сповістити адміністратора про можливу неполадку.

Подібні умови можуть перевірятися в різних частинах системи. Фронтальні сервери можуть контролювати статус з’єднання з пристроями. Підсистема обробки даних може періодично (за розкладом) робити запит до сховища операційних даних задля того, аби виявити потенційно некоректні дані, спричинені неправильною роботою пристрою.

Опишемо також передачу команд від адміністратора до пристрою. Команда від адміністратора надходить до фронтального сервера, який надійно зберігає її в базі адміністративних даних.

Пристрої періодично опитують фронтальні сервери на предмет появи нових команд. Потоки даних показано на рис. 2.4.

Така архітектура потрібна для того, щоби надійно доправляти дані до пристроїв, бо вони можуть бути тимчасово недоступні (відключені або не мати з’єднання з сервером).

Узагалі, зменшити затримку передачі даних і зекономити ресурси можна було б, доправляючи дані безпосередньо (в разі можливості). Скажімо, якщо фронтальний сервер має відкрите з’єднання з пристроєм (скажімо, по WebSocket), то при надходженні команди від адміністратора можна відправляти її відразу. Але це б зменшило гарантії доставки. Якщо команду записано в базу, то пристрій може отримати її, виконати, і лише після цього направити підтвердження фронтальному серверу, який відмітить команду як

прийнято або видалено. Якщо пряме керування командою завершено, воно також може підтвердити його збереження, але якщо під час звільнення виникла помилка, неможливо буде запитати цю команду на передній панелі сервера, але це буде не буде збережено. Якщо застосувати підтвердження не відхилення команд до виконання цього, то адмініструвати доводиться до кінця звернення до запиту, що буде поширюватися на інтерференцію з асинхронною хімією, що підтверджується негайно підтвердженою, оскільки зберігається в базі.

Тепер, скажімо, що відбувається, коли ви отримуєте аналітичні запити. Фронтальні автори аутентифікують запит і визначають його тип. Запит може бути перепризначений до наявних даних або вимагати додаткових витрат.

Доступними є дані, які з'явилися з основним прапорцем і зберігаються в робочій базі даних. Для цього типу запиту слід читати та дублювати дані.

Якщо запитується однаковий аналітичний запит, то має бути можливим визначити план його вилучення, прочитати дані з іншого місця проживання бази даних, а потім утриматися від нього. Якщо доступна лише додаткова плата за доступні дані, на вхід контрольного списку надається інформація тільки з бази даних операційних даних. База даних адміністративних даних може використовуватися для перевірки, чи вимагає запит читання певної інформації про продукт. Скажімо, аналітикам відомо агрегування даних від конкурентів у певному місті. У цьому випадку, першим кроком є пошук адміністративних даних в базі даних адміністративних даних в meccenocculi, а потім зчитування оперативних даних від відповідних конкурентів для покинутих ідентифікаторів.

2.2 Протоколи і формати обміну даними між компонентами системи

Опишемо протоколи передачі даних між компонентами системи.

Зв’язок між пристроями (або шлюзами) і платформою відбувається по зашифрованому протоколу HTTP/2, який використовує криптографічний протокол TLS 1.2. Шифрування трафіку потрібне, аби уникнути перехоплення даних та інших атак. Протокол HTTP/2, стандартизований у 2015 році, дуже добре підходить для IoT, бо:

- Розроблений як оптимізація HTTP/1.1;
- Реалізує стискання HTTP-заголовків (HPACK) за допомогою коду Хаффмана. Цей метод має високу ефективність, але потребує мало пам’яті, що дуже важливо для малопотужних пристроїв;
- За допомогою таких технік як pipelining та мультиплексування заохочує використання одного TCP-з’єднання, що дозволяє уникати повільного тристороннього рукостискання (three-way handshake), яке, до того ж, потребує додаткових ресурсів. У зашифрованому протоколі перевикористання з’єднання дозволяє уникнути ще більш довгого TLS діалогу (negotiation);
- Передбачає операцію PING на рівні протоколу, що дозволяє використовувати її для перевірки працездатності з’єднання з пристроєм;
- Підтримує високорівневу сумісність із HTTP/1.1: методи, коди стану, URI, поля заголовків.

Багато платформ Інтернету речей використовують MQTT, який є набагато ефективнішим за HTTP/1.1. Проте із появою HTTP/2 за продуктивністю HTTP/2 і MQTT майже зрівнялися.

Головною перевагою використання HTTP є його поширеність і загальноприйнятість у Web, що означає наявність функціональних і стабільних серверних і клієнтських бібліотек.

HTTP/2 реалізує техніку server push, яка дозволяє серверу ініціювати відправку даних клієнту замість традиційної моделі «запит-відповідь». Проте

більш природнім і ефективним для повнодуплексного зв’язку є застосування протоколу WebSocket. Пристрої, якщо вони мають можливість установити з’єднання з сервером, підтримують відкритий канал, по якому можуть отримувати команди адміністраторів від сервера за протоколом WebSocket. Використовується захищений варіант протоколу WebSocket – WebSocket Secure (WSS).

У незвичайному вигляді файл даних, в якому фронтальні сервери отримують інформацію, використовується JSON. Він, як і HTTP, дуже поширенний; може читатися людиною (вона не двійковим); є інструментом регулювання та серелізації.

Перевірки, але їх можна розраховувати за рахунок стандартного методу обв'язки, наприклад gzip.

Загалом, REST API реалізує на фронтальних серверах, які можуть бути використані клієнтами, адміністраторами та аналітиками.

Як вже зазначалося, зв'язок між сусідніми частинами і тим самим переходить в хмарні структури платформи, що має місце у випадку використання зашифрованого з'єднання. Припинення SSL знаходиться на рівні балансування навантаження, а фронтальні вуглеводи містять вже відкриті дані. Можливо, хочеться звернутись до зв'язку з центром обробки даних, який ви бажаєте вважати безпечним: трафік здійснюється або ізолює від інших клієнтів, які використовують або допомагають інтелектуальною мережею мережі. Для цього він також дозволяє витягувати дешифрування для більш конкретних, більш про- дуктивних рішень, а у фронтальних агентствах проводити рекурсію тільки за стандартною логікою запитів.

Передача даних між наступними поданнями, контрольним списком даних, робочою базою даних, адміністративною базою даних і фронтальними мозаїками знаходяться в деяких бінарних документах через TCP, які реалізуються відповідними бібліотеками. Наприклад, використовується перевірка даних

Схему передачі формату даних та протоколів між компонентами можна побачити на рис. 2.6.

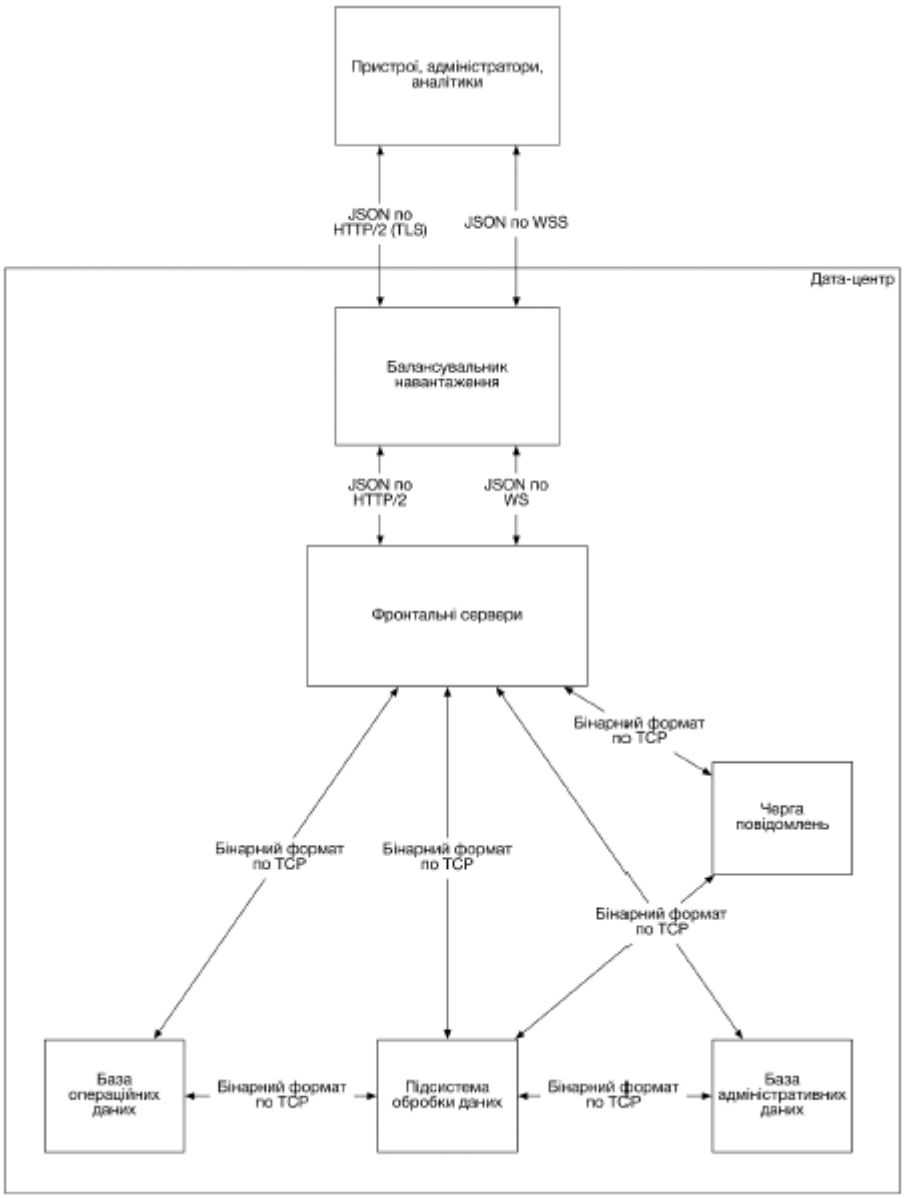

Рисунок 2.6 – Формати даних і протоколи передачі між компонентами

2.3 Мікросервісна архітектура

Так само, як не існує формального визначення терміну мікросервіси, не існує стандартної моделі, яку ви побачите в кожній системі на основі цього архітектурного стилю. Але можна очікувати, що більшість мікросервісних систем мають декілька помітних характеристик.

Шість характеристик мікросервісів:

1) Множина компонентів. Програмне забезпечення, побудоване у вигляді мікросервісів, за визначенням може бути розбито на декілька компонентних послуг. Чому? Таким чином, кожна з цих служб може бути розгорнута, налаштована, а потім розгорнута самостійно без шкоди для цілісності програми. Як наслідок, вам може знадобитися лише змінити один або більше різних служб, замість того, щоб повторно розгортати цілі програми. Але цей підхід має свої недоліки, включаючи дорогі віддалені виклики (замість дзвінків у процесі роботи), більш грубі віддалені API, а також підвищену складність при перерозподілі відповідальності між компонентами.
2) Побудований для бізнесу. Стиль мікросервісів, як правило, організований навколо бізнес-можливостей і пріоритетів. На відміну від традиційного монолітного підходу до розвитку, де різні команди мають особливий акцент на, скажімо, інтерфейсах користувача, базах даних, технологічних шарах або серверній логіці, архітектура мікросервісу використовує міжфункціональні команди. Обов'язки кожної команди полягають у створенні конкретних продуктів на основі однієї або декількох індивідуальних послуг, що спілкуються через шину повідомлень. У мікросервісах команда володіє продуктом протягом свого життя, як і в часто цитованій максимі Амазонки «Ви будуєте її, ви її запускаєте.

3) Проста маршрутизація. Мікросервіси подібні до класичної системи UNIX: вони отримують запити, обробляють їх і відповідно генерують відповідь. Це протилежно тому, як багато інших продуктів, таких як ESB (Enterprise Service Buses) працюють, де використовуються високотехнологічні системи для маршрутизації повідомлень, хореографії та застосування бізнес-правил. Можна сказати, що мікросервіси мають розумні кінцеві точки, які обробляють інформацію та застосовують логіку, і дурні труби, через які проходить інформація.
4) Децентралізована. Оскільки мікросервіси передбачають різноманітність технологій і платформ, методи старої школи централізованого управління не є оптимальними. Децентралізована система управління підтримується спільнотою мікросервісів, оскільки її розробники прагнуть створити корисні інструменти, які потім можуть використовуватися іншими для вирішення тих самих проблем. Так само, як і децентралізоване управління, архітектура мікросервісу також сприяє децентралізованому управлінню даними. Монолітні системи використовують єдину логічну базу даних у різних додатках. У додатку мікросервісу кожна служба зазвичай управляє своєю унікальною базою даних.
5) Нестійкість. Як добре округла дитина, мікросервіси призначені для того, щоб впоратися з невдачею. Оскільки кілька унікальних і різноманітних служб спілкуються разом, цілком можливо, що служба може вийти з ладу з тієї чи іншої причини (наприклад, коли постачальник не доступний). У цих випадках клієнт повинен дозволити своїм сусіднім службам функціонувати, коли він виступає якомога граціозно. Однак моніторинг мікросервісів може допомогти запобігти ризику невдачі. З очевидних причин ця вимога додає більше складності до мікросервісів у порівнянні з архітектурою

монолітних систем.

6) Еволюційне. Архітектура Microservices є еволюційним дизайном і, знову ж таки, ідеально підходить для еволюційних систем, де ви не можете повністю передбачити типи пристроїв, які можуть одного дня отримати доступ до вашої програми. Багато додатків запускаються на основі монолітної архітектури, але, як з'явилися кілька непередбачених вимог, їх можна повільно переробити на мікросервіси, які взаємодіють над старою монолітною архітектурою через API.

Приклади мікрослужб:

- Netflix має широко поширену архітектуру, яка перетворилася з монолітної на SOA. Він отримує більше одного мільярда дзвінків щодня, від більш ніж 800 різних типів пристроїв до свого потокового відео API. Кожен виклик API потім запрошує близько п'яти додаткових викликів до сервісу бекенда.

- Amazon також перейшла на мікросервіси. Вони отримують незліченні дзвінки з різних додатків, включаючи програми, які керують API веб-сервісу, а також сам веб-сайт, що було б просто неможливо для їхньої старої, дворівневої архітектури.

- Аукціонний сайт eBay є ще одним прикладом, який пройшов через той самий перехід. Їх основний додаток складається з декількох автономних додатків, кожен з яких виконує бізнес-логіку для різних областей функцій.

2.4 Підсистема прийому операційних даних. Організація, вимоги та обґрунтування вибору стороннього рішення

Крім того, можна використовувати швидку швидкість, а також змусити вас грати в трафіку, збираючи, перш ніж отримувати дані на - лобовому рівні для балансування навантаження. Для турецької архітектури платформа забезпечує пористість чорних підлог.

Черги решти задаються синхронним косметичним протоком. Це означає, що виправдач і одержувач допомоги не поширюються один на одного. Побудовані, як правило, в черзі, зберігаючи їх, поки вони не отримані.

Важливо, щоб черги субкультури забезпечували впровадження кіберзлочинності та корисного навантаження для підтримки різних можливостей зберігання даних. Для зменшення надмірно віддалених тропічних лісів їх збільшення засноване на їхній інформації. У довгостроковій перспективі, якщо програма приймача є або чергою подань на підтримку роботи в результаті зіткнення, то пошкодження буде безпечним і буде адресовано одержувачам, вся система зможе працювати.

При виконанні черги, отримуйте підтримку, стежте за частішими поїздками. Загалом, забезпечення наступного підходу архівних установ вдалося залучити велику кількість інтересів.

Ми моделюємо такі черги, які слід враховувати при виборі існуючих рішень:

1) Газоподібне паливо. Зробити нову запис у черзі;
2) Горизонтальний маестро. Слідкуйте за кандидатами, які можуть допомогти вам збільшити кількість співробітників;
3) Можливість дотримуватися диктування. Збір, зменшення кількості втрачених даних

Збудження Це саме той випадок, коли зчитування даних з великих інтервалів - відтепер, після відкриття п'ятикутного диска, можна втратити велику кількість даних;

4) Реплікація. Для певного збільшення гарантії зберігання даних і зберігання dotsproptie черги записів для одержувачів при з'єднанні глав серверів;
5) Можливість конфігурування рівня підтримки для відмови. Тобто, оскільки представники компанії, реплікація і час виходу на пенсію зберігаються в DCMC. Візьміть його, щоб переконатися, що у вас є перевага, якщо у вас є великий особистий дохід. З великим завантаженням, щоб бути впевненим, що ви можете отримати гарантію;
6) Може перешкодити групі розповсюдити на теми. Таким чином, ви можете використовувати одержувачів для аудиту для певної ваги, щоб зупинити її, наприклад, відмовитися від вкладення, в якому ви хочете йти.

Найкращими каналами повернення Apache Kafka є. Ми виправдовуємо цей вибір.

2.4.1.1 Apache Kafka – черга повідомлень для прийому операційних даних

У Big Data використовується величезний обсяг даних. Що стосується даних, то ми маємо дві основні проблеми. Перший виклик полягає в тому, як зібрати великий обсяг даних, а другий - проаналізувати зібрані дані. Щоб подолати ці виклики, вам потрібна система обміну повідомленнями.

Kafka призначена для розподілених високопродуктивних систем. Кафка прагне працювати дуже добре, як заміна більш традиційного брокера повідомлень. У порівнянні з іншими системами обміну повідомленнями, Kafka має кращу пропускну здатність, вбудовану роздільну здатність, реплікацію і властиву відмовостійкості, що робить його придатним для масштабних програм обробки повідомлень.

Що таке система обміну повідомленнями?

Система обміну повідомленнями відповідає за передачу даних з однієї програми до іншої, тому програми можуть зосереджуватися на даних, але не турбуватися про те, як поділитися нею. Розподілений обмін повідомленнями заснований на концепції надійного чергування повідомлень. Повідомлення ставляться в чергу асинхронно між клієнтськими програмами та системою обміну повідомленнями. Доступні два типи шаблонів обміну повідомленнями - один - точка-точка, а інша - система обміну повідомленнями-підписка (pub-sub). Більшість моделей обміну повідомленнями слідують за публікацією публікації.

Система обміну повідомленнями з точки до точки

У системі точка-точка повідомлення зберігаються в черзі. Один або більше споживачів можуть споживати повідомлення в черзі, але певне повідомлення може споживатися лише одним споживачем. Як тільки споживач прочитає повідомлення в черзі, він зникає з цієї черги. Типовим прикладом цієї системи є система обробки замовлень, де кожне замовлення обробляється одним процесором замовлень, але процесори з кількома замовленнями можуть працювати одночасно. Наступна діаграма зображує структуру.

система обміну повідомленнями "точка-точка"

Система публікації-підписки

У системі публікації-підписки повідомлення зберігаються в темі. На відміну від системи точка-точка, споживачі можуть підписатися на одну або кілька тем і споживати всі повідомлення в цій темі. У системі Publish-Subscribe виробники повідомлень називаються видавцями, а споживачі повідомлень називаються абонентами. Прикладом реального життя є телевізор Dish, який публікує різні канали, такі як спорт, фільми, музика і т.д., і кожен може підписатися на власний набір каналів і отримувати їх, коли доступні підписані їм канали.

Система публікації-підписки

Що таке Кафка? Apache Kafka є розподіленою системою обміну повідомленнями підписки та надійною чергою, яка може обробляти великі обсяги даних і дозволяє передавати повідомлення з однієї кінцевої точки до іншої. Kafka підходить як для автономного, так і для онлайнового споживання повідомлень. Повідомлення Kafka зберігаються на диску і реплікуються в кластері для запобігання втраті даних. Кафка побудована на базі служби синхронізації ZooKeeper. Він дуже добре інтегрується з Apache Storm та Spark для аналізу потокових даних у реальному часі.

Переваги

Нижче наведено кілька переваг Kafka -

- Надійність. Kafka розподіляється, розділяється, реплікується і відмовляється.
- Масштабованість. Система обміну повідомленнями Kafka легко масштабується без прострочення часу.
- Довговічність. Kafka використовує журнал розподіленої комісії, що означає, що повідомлення зберігається на диску якомога швидше, тому він довговічний.
- Продуктивність. Kafka має високу пропускну здатність як для публікації, так і для підписки на повідомлення. Він підтримує стабільну продуктивність, навіть багато ТБ повідомлень зберігаються.

Кафка дуже швидка і гарантує нульове час простою і нульову втрату даних.

Використовуйте футляри

Кафка може використовуватися в багатьох випадках використання. Деякі з них наведені нижче -

- Метрики. Кафка часто використовується для оперативних даних моніторингу. Це включає агрегування статистики від розподілених додатків для виробництва централізованих каналів оперативних даних.

- Рішення агрегації журналу. Kafka може використовуватися в організації для збору журналів з декількох служб і зробити їх доступними у стандартному форматі для декількох споживачів.

- Потокова обробка. Популярні фреймворки, такі як Storm і Spark Streaming, читають дані з теми, обробляють її і записують оброблені дані до нової теми, де вона стає доступною для користувачів і додатків. Міцна довговічність Кафки також дуже корисна в контексті обробки потоку.

Необхідність Кафки. Kafka - це уніфікована платформа для обробки всіх даних у реальному часі. Kafka підтримує доставку повідомлень з низькою затримкою і дає гарантію на відмовостійкість в разі відмови машини. Він має можливість обробляти велику кількість різноманітних споживачів. Кафка дуже швидка, виконує 2 мільйони записів / сек. Кафка зберігає всі дані на диску, що по суті означає, що всі записи йдуть в кеш сторінки ОС (ОЗУ). Це робить його дуже ефективним для передачі даних з кешу сторінок в мережевий сокет.

В Kafka є такі режими гарантій того, що повідомлення дійдуть до адресата:

– *Максимум один раз (at most once)* – повідомлення можуть бути втрачені, але ніколи не доставляються повторно;

– *Як мінімум один раз (at least once)* – повідомлення ніколи не втрачаються, але можуть бути доправлені повторно;

– *Точно один раз (exactly once)* – повідомлення доставляється точно один раз.

Рисунок 2.7 показує кластерну схему Кафки.

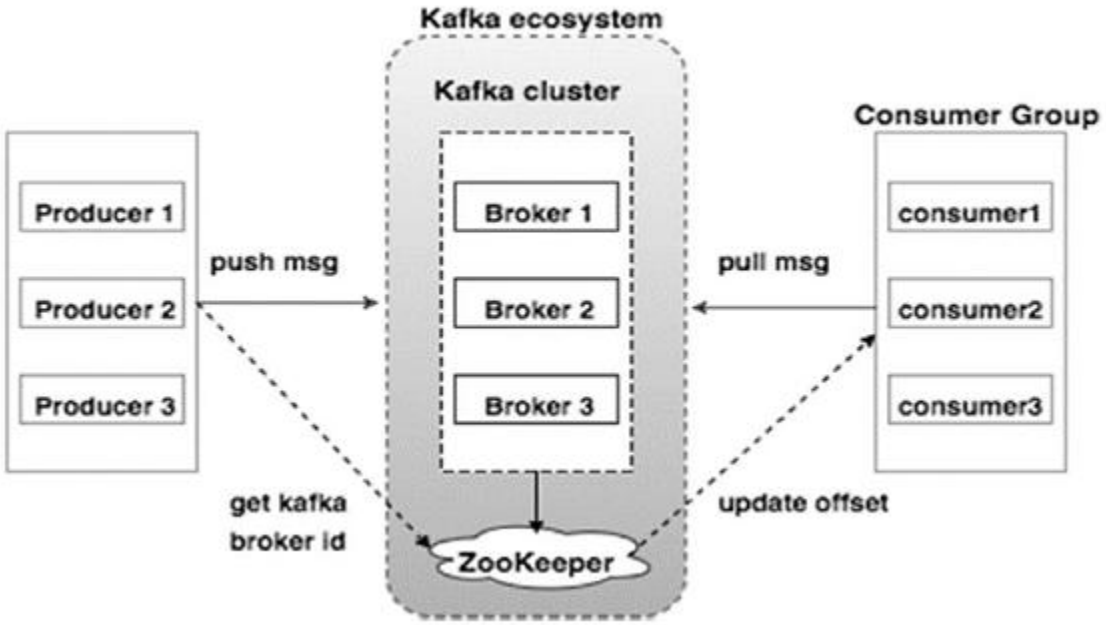

Рисунок 2.7 – кластерна система Kafka

Наступні пункти описують кожен з компонентів, показаних на схемі вище:

1) Брокер. Кластер Кафка зазвичай складається з декількох брокерів для підтримки балансу навантаження. Брокери Kafka не мають стану, тому вони використовують ZooKeeper для підтримки свого кластерного стану. Один екземпляр брокера Kafka може обробляти сотні тисяч зчитувань і записів в секунду, і кожен брокер може обробляти терабайти повідомлень без впливу на продуктивність. Вибір лідера брокера "Кафка" може бути здійснений за допомогою ZooKeeper.

2) ZooKeeper використовується для управління та координації брокера Kafka. Послуга ZooKeeper в основному використовується для повідомлення виробника та споживача про наявність будь-якого нового брокера в системі Кафка або провал брокера в системі Кафка. Відповідно до повідомлення, отриманого Zookeeper про наявність або невиконання брокера, то прокурор і споживач приймають рішення і починають координувати свої завдання з іншим брокером.

3) Виробники Виробники надають дані брокерам. Коли новий брокер запускається, всі виробники шукають його і автоматично відправляють повідомлення цьому новому брокеру. Виробник Kafka не чекає на підтвердження від брокера і надсилає повідомлення так швидко, як може працювати брокер.

4) Споживачі Оскільки брокери Kafka не мають громадянства, це означає, що споживач повинен підтримувати, скільки повідомлень було спожито за допомогою зміщення розділів. Якщо споживач визнає певне зміщення повідомлення, це означає, що споживач спожив всі попередні повідомлення. Споживач видає брокеру асинхронний запит тягнути, щоб мати буфер готових до споживання байт. Споживачі можуть перемотати назад або переходити до будь-якої точки розділу просто шляхом введення значення зміщення. Значення зміщення споживача повідомляється ZooKeeper.

2.5 Підсистема обробки даних. Організація, вимоги та обґрунтування вибору стороннього рішення

Описана в підрозділі «Архітектура платформи» схема обробки даних реалізує так звану **лямбда-архітектуру** (lambda architecture). Розглянемо цей концепт детальніше.

Ламбда-архітектура - це архітектура обробки даних, призначена для обробки великих обсягів даних, використовуючи переваги як пакетної, так і потокової обробки. Такий підхід до архітектури намагається збалансувати затримку, пропускну здатність і відмовостійкість, використовуючи пакетну обробку для забезпечення всебічного і точного перегляду пакетних даних, одночасно використовуючи потокову обробку в реальному часі для забезпечення перегляду даних в Інтернеті [11].

Ось як виглядає це з точки зору високого рівня:

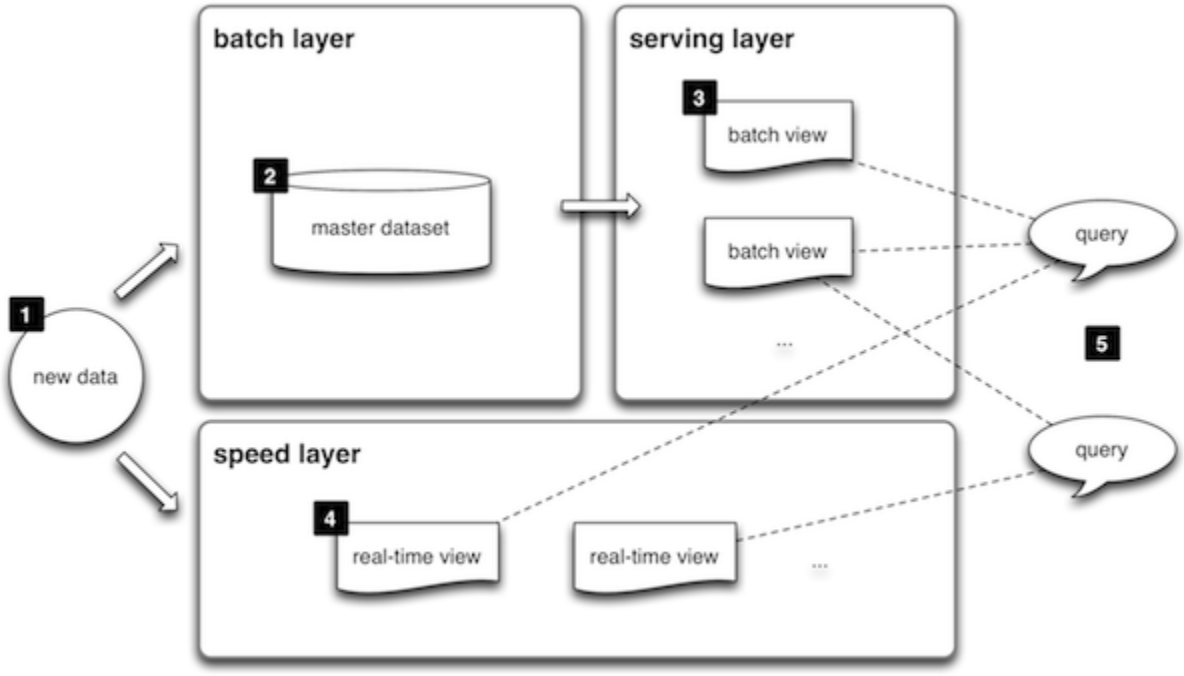

Рисунок 2.8 – структура лямбда-архітектури

1) Всі дані, що надходять у систему, надсилаються як на пакетний шар, так і на швидкісний шар для обробки;
2) Рівень пакетів має дві функції:

- керування магістральним набором даних (незмінним, лише додаванням вихідних даних);
- попереднім обчисленням переглядів пакету.

3) Сервісний шар індексує вигляд пакету, щоб вони могли запитуватися в ad-hoc способі з низькою затримкою.
4) Швидкість швидкості компенсує високу затримку оновлень для обслуговуючого шару і стосується лише останніх даних.
5) На будь-який вхідний запит можна відповісти, об'єднавши результати перегляду пакетів і переглядів у реальному часі.

Опишемо також підхід **MapReduce**, який частково застосовується для обробки даних платформою.

Що таке MapReduce? MapReduce - це технологія обробки та програмна модель для розподілених обчислень на основі java. Алгоритм MapReduce містить дві важливі завдання, а саме: Map і Reduce. Карта приймає набір даних і перетворює її в інший набір даних, де окремі елементи розбиваються на кортежі (пари ключ / значення). По-друге, зменшити завдання, яке приймає вихідні дані з карти як вхідні дані і об'єднує ці дані кортежів у менший набір кортежів. Як випливає з послідовності імені MapReduce, завдання зменшення завжди виконується після завдання на карті.

Основною перевагою MapReduce є те, що легко масштабувати обробку даних на декількох обчислювальних вузлах. Під моделлю MapReduce примітиви обробки даних називаються мапперами і редукторами. Декомпозиція програми обробки даних в маппери і редуктори іноді нетривіальна. Але, коли ми напишемо програму у формі MapReduce, масштабування програми для

запуску на сотні, тисячі або навіть десятки тисяч машин у кластері - це лише зміна конфігурації. Така проста масштабованість привернула багатьох програмістів до використання моделі MapReduce.

Алгоритм

1) Взагалі парадигма MapReduce базується на відправленні комп'ютера туди, де зберігаються дані!
2) Програма MapReduce виконується у три етапи, а саме на етапі карти, етапі перетасовування та зменшенні ступеня.
   - Сценарій Map - робота карти або картографує обробляти вхідні дані. Як правило, вхідні дані у вигляді файлу або каталогу зберігаються у файловій системі Hadoop (HDFS). Вхідний файл передається функції відображення рядка за рядком. Маппер обробляє дані і створює кілька невеликих фрагментів даних.
   - Зменшити стадію - цей етап є комбінацією етапу Shuffle і етапу зменшення. Завдання Reducer - обробляти дані, які надходять з картографування. Після обробки вона виробляє новий набір виводу, який буде зберігатися в HDFS.
3) Під час завдання MapReduce Hadoop надсилає завдання Map і Reduce на відповідні сервери кластера.
4) Рамка керує всіма деталями передачі даних, такими як видача завдань, перевірка завершення завдання та копіювання даних навколо кластера між вузлами.
5) Більшість обчислень відбувається на вузлах з даними на локальних дисках, що зменшує мережевий трафік.
6) Після завершення цих завдань кластер збирає і зменшує дані, щоб сформувати відповідний результат, і відправляє їх назад на сервер Hadoop.

2.5.1 Apache Spark – система обробки даних

Apache Spark є потужним уніфікованим аналітичним механізмом для масштабної розподіленої обробки даних і машинного навчання. На вершині основного механізму обробки даних Spark використовуються бібліотеки для SQL, машинне навчання, обчислення графів і обробка потоків. Ці бібліотеки можуть використовуватися разом на багатьох етапах в сучасних конвеєрах даних і дозволяють повторне використання коду в пакетних, інтерактивних і потокових додатках. Spark корисний для обробки ETL, аналітики та завантаження машинного навчання, а також для пакетної та інтерактивної обробки запитів SQL, висновків машин для навчання та додатків штучного інтелекту.

Значна частина сили Спарка полягає в його здатності поєднувати дуже різні техніки і процеси в єдине, цілісне ціле. За межами Spark, дискретні завдання вибору даних, перетворення даних різними способами і аналіз трансформованих результатів можуть легко вимагати серій окремих фреймворків обробки, таких як Apache Oozie. Spark, з іншого боку, пропонує можливість поєднувати їх, перетинаючи кордони між пакетним потоком, потоковим потоком і інтерактивними робочими процесами таким чином, щоб зробити його більш продуктивним.

Завдання іскри виконують декілька операцій послідовно, в пам'яті, тільки проливаючи на диск, коли це вимагається обмеженнями пам'яті. Spark спрощує управління цими розрізненими процесами, пропонуючи інтегрований комплекс - конвеєр даних, який легше налаштовувати, запускати і підтримувати. У таких випадках, як ETL, ці трубопроводи можуть стати надзвичайно багатими і складними, поєднуючи велику кількість входів і широкий спектр кроків обробки в єдине ціле, що послідовно забезпечує бажаний результат.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Тестування є важливою та невід'ємною частиною продукту надання послуг на ринку хмарних технологій для розумних будівель. Функції тестування програмного забезпечення призначене для використання в цій області є те, що наше місце тут означає коригування всіх компонентів аналізу розрахункового виробу. Тобто, основною метою є отримання достовірних аналітичних і математичних результатів на виході. Важливою особливістю є те, що ринок хмарних розумних технологій дуже великий і молодий. Також важливо індикатор змінюється в часі і на один пристрій. Це пов'язано з тим, що ці різні пристрої мають різну волатильність і різну ціну для всіх Ви повинні створити тест для таких розрахунків. Набір контрольних даних використовується в основному для цілей перевірки. Такі програми мають бути перевірені за допомогою даних різних служб (джерела даних) для різних. Зазвичай дефекти розрахунку виявляються при використанні пошкодженого або штучно спотворені дані. У таких продуктах візуалізація вже є продуктом різних аналітичних розрахунки, тому мені приємно перевіряти точність відображення даних порівнювати ці дані з параметрами, що використовуються користувачем. Окремим етапом тестування є перевірка правильності розвитку всіх функції інтерфейсу користувача.

3.1.1 Компонентне тестування

Методом компонентного тестування будуть перевірені логічно окремі частини веб сервісу, такі як:

- роути сервісу;
- методи доступу до бази даних;
- окремі допоміжні функції;
- методи авторизації.

3.1.2 Інтеграційне тестування

Методом інтеграційного тестування будуть перевірені взаємодії між модулями системи, такі як:

- взаємодія веб сервера з базою даних;
- взаємодія веб сервера з клієнтом.

3.1.3 Тестування продуктивності

Методом тестування продуктивності буде перевірена швидкодія наступних елементів системи:

- запити до бази даних;
- обробка сервером запитів без бази даних;
- обробка сервером запитів з базою даних.

3.2 Критерії проходження тестування

3.2.1 Компонентне тестування

Для компонентного тестування критерієм проходження є успішне виконання кожного пункту тесту. У разі якщо хоча б один

пункт не був успішно виконаний – тестування вважається не пройденим.

3.2.2 Інтеграційне тестування

Для інтеграційного тестування критерієм проходження є успішне виконання кожного пункту тесту. У разі якщо хоча б один пункт не був успішно виконаний – тестування вважається не пройденим.

3.2.3 Тестування швидкодії

Для тестування швидкодії критерієм проходження є успішне виконання тесту з кожним доступним набором параметрів (кількість даних у одному запиті, кількість запитів, конкурентність) не довше ніж максимально допустимий час. У разі якщо хоча б один варіант тесту не був успішним або виконувався довше максимально допустимого часу – тестування вважається не пройденим.

3.3 Процес тестування

3.3.1 Дані до тестів

Вхідними даними для компонентного тестування є набори параметрів на яких очікується певний результат, що є вихідними даними даного тесту.

Вхідними даними для інтеграційного тестування є набори повідомлень що будуть передані від одного компоненту системи до іншого відповідно до конкретного тесту. Вихідними даними для даного виду тестування є результат роботи останнього компоненту у ланцюзі (наприклад, запис у базі даних у випадку тестування взаємодії API серверу та бази даних).

Вхідними даними до тестування швидкодії є набори даних, що покривають усі варіанти роботи системи у конкретному випадку. Вихідними даними є швидкість обробки запитів, кількість оброблених

запитів, кількість неправильних реакцій на набір даних, дані по навантаженню на апаратну платформу (завантаженість процесору, вільна оперативна пам’ять, завантаженість мережі тощо).

3.3.2 Задачі тесту

Кожен тест повинен перевірити як правильність програми у відповідності до умов виконання тесту (test-driven development), так і виявити можливі помилки у роботі.

3.3.3 План виконання

Компонентне тестування повинне виконуватися до інтеграційного, яке, у свою чергу, виконується до тестування швидкодії.

3.4 Вимоги до середовища

3.4.1 Апаратна частина

Вимоги до апаратної частини у технічному завданні.

3.4.2 Програмна частина

Для виконання тестування платформа повинна мати операційну систему на базі Linux.

3.4.3 Вимоги до безпеки

Для тестування бажано створити відповідний тестовий акаунт.

3.4.4 Інструменти

Для виконання тестування використовувати наступні програмні інструменти:

– Insomnia;

– Vegeta (https://github.com/tsenart/vegeta) – для тестування швидкодії.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Для повного розгортання веб серверу необхідно:

– Встановити основний сервіс;

– Встановити СКБД PostgreSQL

– Налаштувати файл WebApp.dll.config

– Здійснити міграцію структури БД в СКБД

–Запустити основний сервіс.

4.1.1 Встановлення основного сервісу

Для встановлення основного сервісу необхідно взяти файли директорії /Bin/Server та помістити в необхідну робочу папку.

4.1.2 Встановити СКБД PostgreSQL

Для встановлення СКБД необхідно перейти на сайт PostgreSQL та завантажити для необхідної платформи. Далі, користуючись інструкцією з встановлення встановити та вибрати порт, стандартного користувача та пароль. Необхідно, щоб доступ до серверу СКБД був з серверу з веб сервісом.

4.1.3 Налаштувати файл WebApp.dll.config

Необхідно налаштувати порт застосунку та дані доступу до бази даних, для цього слід заповнити поля файлу /Bin/WebApp.dll.config, а саме port, dbConnectionString

4.1.4 Здійснити міграцію структури БД в СКБД

Для здійснення міграції слід скопіювати вміст файлу /Sql/Initial.sql та виконати його в IDE для СКБД до потрібної бази даних.

4.1.5 Запустити основний сервіс

Для запуску серверу слід виконати команду dotnet run WebApp.dll до файлу, взятого з папки /Bin/WebApp.dll

4.2 Інструкція користувача

Інструкція наведена в додатку Інструкція користувача

4.3 Інструкція адміністратора

Інструкція наведена в додатку Інструкція системного програміста

ВИСНОВКИ

У ході виконання даного дипломного проекту було досліджено та побудовано платформу для Інтернету речей.

Зокрема, було виділено основні вимоги, що висуваються до подібних систем, вивчено основні підходи до вирішення задач обробки великих за обсягами даних, досліджено існуючі платформи та виокремлено шляхи їх покращення.

Розроблена платформа задовольняє поставленим функціональним і нефункціональним вимогам та містить реалізацію таких підсистем: приймання, зберігання, обробки операційних даних; автентифікації та безпеки; роботи з адміністративними даними; моніторингу. Це дозволяє застосування її такими виділеними групами користувачів: пристроями, адміністраторами, аналітиками і адміністраторами безпеки.

Було запропоновано та реалізовано таку розподілену сервісно-орієнтовану (мікросервісну) архітектуру платформи, яка забезпечує високу продуктивність і майже нескінченну масштабованість. Це дозволяє нарощувати потужність системи для обслуговування мереж пристроїв будь-яких розмірів і для збільшення можливостей підсистеми аналітики даних. Ефективна реалізація компонентів системи дає змогу оптимізувати витрати на апаратне забезпечення платформи.

Платформа також гарантує значний рівень відмовостійкості, що знижує ймовірність втрати операційних даних і забезпечує високий рівень її доступності. Досліджені та реалізовані аспекти безпеки надають достатній рівень захищеності даних при передачі й убезпечують систему від різноманітних інформаційних атак.

У ході розробки платформи були з належним обґрунтуванням відібрані та застосовані сучасні системи, протоколи, формати даних, бібліотеки та бази

даних. Використанні сторонні рішення мають відкритий вихідний код, що спрощує розширення системи, має економічні та інші переваги.

Було детально описано різноманітні аспекти обраних архітектурних рішень: від обґрунтування високорівневого поділу на функціональні модулі до особливостей ефективної реалізації вводу/виводу та впливу обраної схеми автентифікації на масштабованість архітектури платформи.

На відміну від існуючих платформ, розроблене рішення є добре розширюваним у тому сенсі, що його можливо легко адаптувати під аналітику даних, які надходять від різноманітних пристроїв і сенсорів, задаючи ланцюжки обробки даних. Протоколи та формати даних було обрано не лише з огляду на їх ефективність, а ще й враховуючи зручність і простоту розширення платформи.

Іншими ключовими особливостями платформи є моніторинг і функціональність надійної передачі службової інформації між пристроями та адміністраторами. Ці можливості дозволяють ефективно керувати мережею пристроїв, а також гнучко налаштовувати інструменти попередження про можливі неполадки та оперативно на них реагувати.

Запропоновані та застосовані в роботі архітектурні та технічні рішення є достатньо універсальними, що дозволяє використовувати їх для побудови різноманітних систем обробки великих даних та інших платформ для Інтернету речей, спеціалізованих для більш вузьких областей.

Роботу платформи було показано за допомогою демонстраційного прикладу, в якому моделюється робота системи з мережею метеорологічних станцій. Також до платформи було підключено мікрокомп’ютер Raspberry Pi 3 та продемонстровано обробку даних із його сенсора. Ці приклади охоплюють більшість можливостей системи та показують її застосування всіма групами користувачів. Описано реалізацію на боці клієнта процесів реєстрації та автентифікації пристрою, а також організацію відправки даних до сервера.

ПЕРЕЛІК ПОСИЛАНЬ

1. Доповідь компанії Gartner. – Режим доступу: http://www.gartner.com/newsroom/id/3165317. – Дата доступу: 23.05.2016.

2. Доповідь компанії Cisco. – Режим доступу: http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoE_Economy.pdf. – Дата доступу: 23.05.2016.

3. Nathan Marz. Big Data: Principles and best practices of scalable realtime data systems / Nathan Marz, James Warren. – Manning Publications, 2015. – 328 p.

4. Arvind Sathi. Big Data Analytics: Disruptive Technologies for Changing the Game / Arvind Sathi. – Mc Press, 2012. – 96 p.

5. Apache Spark documentation - https://spark.apache.org/documentation.html

6. Apache Hadoop documentation - https://hadoop.apache.org/docs/stable/

7. Microservices architecture - https://microservices.io/patterns/microservices.html

8. Internet of Things – Architecture IoT-A Deliverable D1.5 – Final architectural reference model for the IoT v3.0 - https://iotforum.org/wp-content/uploads/2014/09/D1.5-20130715-VERYFINAL.pdf

9. Securing Internet of Things (IoT) with AWS - https://d1.awsstatic.com/whitepapers/Security/Securing_IoT_with_AWS.pdf

10. IoT from cyber security perspective Case study JYVSECTEC - https://www.theseus.fi/bitstream/handle/10024/151498/IoT%20from%20cyber%20security%20perspective.pdf?sequence=1&isAllowed=y

11. Lambda architecture – how to build a big data pipeline – https://towardsdatascience.com/lambda-architecture-how-to-build-a-big-data-pipeline-part-1-8b56075e83fe